ソフトウェア工学 140－6
（２００３．３．６）

# 並行オブジェクトモデルから並行スレッドモデルへの変換法

岡崎光隆[†]　青木利晃[†,††]　片山卓也[†]

オブジェクトを単位としたモデルは大規模・複雑なシステムの分析に広く利用されている．一方，スレッドを単位としたモデルは，実時間性を考慮したシステムの設計に役立つモデルである．組み込みソフトウェアのように，大規模かつ高い実時間性が求められるシステムに対して，これらの両モデルを使用した開発法が提案されている．しかし，モデル間に厳密な対応が付けられていないため，モデルの相互変換ができないという問題がある．我々はこれまでに，並行正規表現を用いてモデル間の対応関係を形式的に定義し，公理系を用いてモデルを変換する手法を提案した．本稿では，この変換を自動的に実行するツールを提案し，例題モデルの変換に成功した事例を紹介する．

# Transforming Concurrent Objects into Concurrent Threads

MITSUTAKA OKAZAKI,[†] TOSHIAKI AOKI[†,††] and TAKUYA KATAYAMA [†]

Object-oriented models are widely used in large software system developments. On the other hand, thread-based models are useful for handling real-time constraints of systems. Recently, these two kind of models have been adopted to embbeded system developments because of the size of the systems and severe real-time constraints. However, existing development methods do not clarify the relationship of these models. It is hard to transform the object-oriented model into the thread-based model during development. For this problem, we have proposed a model transformation method using Concurrent Regular Expressions. In this paper, we introduce a support tool for automatic transformation of the models using the method.

## 1. はじめに

近年の組み込みソフトウェア開発では，厳しいパフォーマンスの要求に加え，ソフトウェアの大規模・複雑化という問題への対応が求められるようになってきた．この問題に対し，分析工程に既存のオブジェクト指向技術を適用し，設計工程以降では処理列中心の設計手法を利用する開発法が提案されている[2)3)]．大規模かつ複雑なシステムのモデル化には，オブジェクト指向分析が高い実績を持つ．一方，高い実時間性が要求されるソフトウェアの開発には，タスクやスレッドなど，システム内で行われる処理列を単位とした設計技術が有効である．

ここで，処理列とは，システムのある機能が実行されたときに，オブジェクト間に発生する通信の列を意味している．オブジェクト中心の観点では，システムの個々の機能は複数のオブジェクトの協調動作により実現されるものである．システムのある機能が実行される過程は，オブジェクト間に発生した通信の列として観測される．このような通信の列を本稿ではスレッドと呼び，設計モデルの構成単位として考える．図 1 にオブジェクトとスレッドの対応関係を示す．実時間制約は一般に，スレッドの応答時間の制約として与えられる．

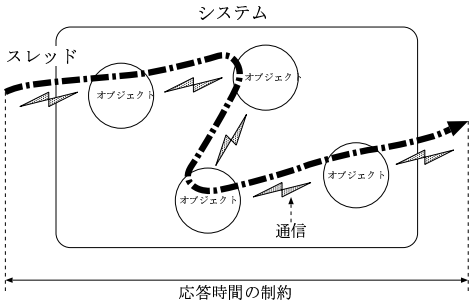



**図 1**　オブジェクトとスレッド

システムの設計工程では，実時間制約を満足するため，頻繁にモデルの修正が行われる．ここで，モデル単位がオブジェクトである場合，ある機能の実時間制約を満たすための修正が，その機能に関連した全てのオブジェクトに波及する可能性がある．場合によっては，オブジェクトの分解や結合など，モデルの構造を破壊する変更が起りうる．一方，モデル単位がスレッ

† 北陸先端科学技術大学院大学 情報科学研究科
Japan Advanced Institute of Science and Technology
†† 科学技術振興事業団 さきがけ 21
PRESTO, Japan Science and Technology Corporation

ドである場合，このような変更はモデル単位の内部に閉じるので，容易かつ安全に変更を扱うことができる．

設計工程でスレッドを単位としたモデル化を行うためには，分析工程でモデル化されたオブジェクトの振る舞いを解析し，対応する設計モデルのスレッドを抽出する手法が必要になる．しかし，これまでに提案されている開発法では，オブジェクトとスレッドの厳密な対応関係が明らかになっておらず，分析モデルから設計モデルへの系統的な変換ができないという問題があった．この問題に対し我々は，並行正規表現[1] を用いて，オブジェクト中心のモデルからスレッド中心のモデルへ変換する手法を提案した[5]．この変換法は並行正規表現を等価変換する公理系によって形式的に定義されている．しかし，この手法は変換にかかる作業ステップ数が多く，手作業による変換には限界がある．

そこで本研究では，この変換作業を計算機を用いて自動化するシステムを提案する．本稿では，我々の実装した自動変換システムを，メディアプレイヤーの例題仕様に適用し，モデルの自動変換に成功した事例を紹介する．

本稿の構成は次の通りである．まず 2 章で，並行正規表現の定義を紹介する．3 章では並行正規表現を用いて各モデルを定義する．4 章ではモデル変換の手法を述べる．5 章で，メディアプレイヤーのモデルに対する変換法の適用例を紹介する．6 章で我々の実装した自動変換システムを紹介し，自動変換を行った結果を示す．7 章で適用実験の結果について考察し，8 章で本稿を統括する．

## 2. 並行正規表現

並行正規表現は並行システムの振る舞いの代数的な表現として提案された記法で，正規表現に対し 4 つの演算子を拡張したものである．本稿では，それらの演算子のうち，インターリーブ演算子と同期複合演算子の 2 つのみを使用する．

$\Sigma$ を記号の有限集合とすると，$\Sigma$ 上の並行正規表現は以下のように帰納的に定義される．

**定義 2.1 (並行正規表現)**
$c \in \Sigma \cup \{\perp, \epsilon\}$ は並行正規表現である．$P$ と $Q$ を共に並行正規表現，$S$ を記号の集合とした時，$P + Q, P.Q, P^*, P \| Q, P\,[S]\,Q$ および $(P)$ もまた並行正規表現である．

ここで，$\perp$ は空集合を，$\epsilon$ は空列を意味する記号である．任意の並行正規表現 $P$ に対し，$P.\perp = \perp.P = \perp$，$P + \perp = P$ および $P.\epsilon = \epsilon.P = P$ が成立する．

各演算子の結合優先度は $*, ., +, \|, [S]$ の順で高いものとし，適宜括弧を省略して表現する．例えば，$a^*.b\,[S]\,b + c$ は $((a^*).b)\,[S]\,(b + c)$ を意味する．

$\|$ をインターリーブ演算子，$[S]$ を同期複合演算子と呼ぶ．並行正規表現から $\|$ と $[S]$ を除いた表現は従来の正規表現と等価である．並行正規表現に登場する演算子は，直感的には，$+$ が処理の選択，$.$ が処理の連続，$*$ が処理の 0 回以上の繰り返しをそれぞれ意味している．また，$[S]$ は 2 つのオブジェクトが通信を行いながら並行動作することを意味し，$\|$ は 2 つのスレッドが並行動作することを表現する演算子である．なお，並行正規表現本来の同期複合演算子は $[\ ]$ であるが，本稿では，記号の集合を表現する $S$ を拡張した $[S]$ を用いる．この定義では $[\phi]$ が従来の $[\ ]$ と等価な意味を持つ．以降 $[\ ]$ を $[\phi]$ の省略形として用いる．

並行正規表現の厳密な意味は記号列の集合で定義される．並行正規表現 $P$ の意味は並行正規表現から意味への写像 $L$ を用いて $L(P)$ と表記される．$L(P)$ は $P$ の言語とも呼ばれる．本稿では記号列を表現する際に個々の記号の区切りに $\cdot$ を用いる．例えば，$a \cdot b \cdot c$ は $\Sigma = \{a, b, c\}$ 上の記号列の表現の例である．

**定義 2.2 (並行正規表現の言語)**
$c$ を記号の集合 $\Sigma$ の要素とする．また，$P$ および $Q$ を $\Sigma$ 上の並行正規表現とし，$w, x$ および $y$ を $\Sigma$ 上の記号列とする．$S$ は記号の集合とする．このとき，$L$ は以下のように定義される．

$L(\perp) = \phi, L(\epsilon) = \{\epsilon\}, L(c) = \{c\}$
$L(P.Q) = \{x \cdot y \mid x \in L(P),\ y \in L(Q)\}$
$L(P + Q) = L(P) \cup L(Q)$
$L(P^*) = \bigcup_{i=0,1,\ldots} L(P^i)$
$L((P)) = L(P)$
$L(P \| Q) = \{w \mid x \in L(P), y \in L(Q), w \in intl(x \| y)\}$
$L(P\,[S]\,Q) = \{w \mid w \in (\Sigma_P \cup \Sigma_Q)^*,\ w/(\Sigma_P \cup S) \in L(P),\ w/(\Sigma_Q \cup S) \in L(Q)\}$

ここで，$\Sigma^*$ は $\Sigma$ 上の全ての記号列の集合を意味する．また，$\epsilon$ は長さ 0 の記号列を意味し，いかなる $x$ に対しても $\epsilon \cdot x = x \cdot \epsilon = x$ が成立する．$\Sigma_P$ は $P$ に出現する全ての記号の集合を意味する．ただし，$\epsilon$ と $\perp$ は含まない．$P^i$ は $P$ の $i$ 回の連続を意味する．すなわち $P^1 = P, P^2 = P.P, P^3 = P.P.P \cdots$ となる．ただし $P^0$ は常に $\epsilon$ であるとする．$w/\Sigma$ は $w$ に登場する記号を $\Sigma$ に登場する記号に制限した記号列を表現する．言い換えると，$\Sigma$ に登場しない全ての記号が $w$ から削除される．$intl$ は 2 つの記号列のインターリーブ集合を求める関数である．並行正規表現では，ある 2 つの動

作が並行に行われるという事実を，どちらの動作が先に発生してもよいという事実と同じであると考える．このような並行性のモデル化の下で，並行に動作する2つの列の振る舞いの可能性を全て網羅する関数がインターリーブ関数 *intl* である．*intl* の定義は以下の通り．

**定義 2.3 (インターリーブ)**
$intl(\mathtt{a} \,\|\, \epsilon) = intl(\epsilon \,\|\, \mathtt{a}) = \{\mathtt{a}\}$
$intl(\mathtt{a} \,\|\, \mathtt{b}) = \{\mathtt{a}\cdot\mathtt{b},\ \mathtt{b}\cdot\mathtt{a}\}$
$intl(\mathtt{a.x} \,\|\, \mathtt{b.y}) =$
$\{\mathtt{a}\cdot\mathtt{w} \mid \mathtt{w} \in intl(\mathtt{x}\|\mathtt{b.y})\} \cup \{\mathtt{b}\cdot\mathtt{w} \mid \mathtt{w} \in intl(\mathtt{a.x}\|\mathtt{y})\}$
ただし，$\mathtt{a}, \mathtt{b}$ は単一の記号，$\mathtt{w}, \mathtt{x}, \mathtt{y}$ は記号列である．

## 3. モデルの形式化

この節では，並行正規表現を用いて，並行オブジェクトモデルと並行スレッドモデル，および個々のオブジェクトとスレッドの振る舞いを定義する．

並行オブジェクトモデルは，相互に通信を行いながら並行動作するオブジェクトの集合を表現したモデルである．このモデルはシステムの分析モデルに相当し，[ ] 演算子を用いて定義する．なお，[S] の S は4節に示す変換手法で利用されるが，モデルの定義においては常に空集合が与えられるので，この節では S を略して表記する．一方，並行スレッドモデルは，システムの設計モデルに相当し，複数のスレッドが並行動作するようなシステムを表現するモデルである．ただし，各スレッドの間には通信が発生しないことを仮定する．このモデルは ‖ 演算子を用いて定義する．

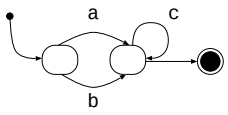
個々のオブジェクトとスレッドは内部的な並行性を持たないことが仮定される．その振る舞いは +,. および * の3種類の演算子を用いて定義する．

### 3.1 オブジェクトとスレッド

例えば，$(\mathtt{a}+\mathtt{b}).\mathtt{c}^*$ はオブジェクト，あるいはスレッドの振る舞いの定義である．記号 a, b, c はそれぞれオブジェクトやスレッドで行われる原始的な動作に対応する．図2にこの定義とこれに対応する状態遷移図を示す．並行正規表現の各記号は状態遷移図の遷移ラベルと対応する．図2が表現する振る舞いは次の通りである．まず始めに a または b が実行される．次に c が任意の回数実行され，動作が終了する．

### 3.2 並行オブジェクトモデル

並行オブジェクトモデルは，複数のオブジェクトが相互に通信を行いながら動作した場合の全体の振る舞いを定義するモデルである．個々のオブジェクトの振る舞いを正規表現で定義し，オブジェクト間の通信と並行動作の表現に [ ] 演算子を用いる．システムに n 個のオブジェクトが存在する時，並行オブジェクトモデルを以下の式で定義する．



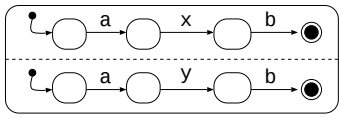
(a+b).c*

**図 2** 振る舞いの定義例

$$\mathtt{O}_1[\ ]\mathtt{O}_2[\ ]\cdots[\ ]\mathtt{O}_n$$

ここで，$\mathtt{O}_1 \cdots \mathtt{O}_n$ は個々のオブジェクトの振る舞いを表現する正規表現である．なお，[ ] には結合法則が成り立つので，ここでは，式の括弧を省略している．

図3は，2つのオブジェクトからなるシステムの例である．各オブジェクトの振る舞いの定義はそれぞれ a.x.b および a.y.b である．図3において，点線で区切られた箱の中にある2つのネストした状態遷移図は，それぞれが並行に動作することを意味している．



a.x.b[ ]a.y.b

**図 3** 並行オブジェクトモデルの例

[ ] 演算子の両辺で共通する記号は，同期通信を意味する記号として扱われる．ここで，同期通信とは以下の制約を満たす通信である．

- 通信に参加したオブジェクトの動作は，通信が完了するまでブロックされる．
- 通信は決して無視されない．

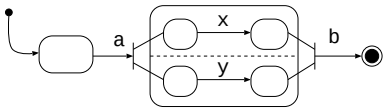
図3のシステムでは，まず始めに2つのオブジェクト間で a が同期して発生する．次に x と y が並行に行われ，最後に再び b が同期して行われる．

### 3.3 並行スレッドモデル

並行スレッドモデルは，並行動作する処理列を含むシステム全体の振る舞いを表現するモデルである．このモデルは正規表現とインターリーブ演算子 ‖ を用いて定義される．図4は並行スレッドモデルの例である．この並行スレッドモデルが表現する振る舞いは，図3の並行オブジェクトモデルと等しい．

## 4. モデル変換手法

並行オブジェクトモデルを表現する式を O と表記し，




a.(x||y).b

**図 4** 並行スレッドモデルの例

O に対応する並行スレッドモデルを T と表記する．この時，2 つのモデルが同じシステムを表現しているならば，両式の意味する振る舞いは一致すべきである．したがって，$L(O) = L(T)$ が成立する．並行オブジェクトから並行スレッドへのモデル変換は，このような関係を満たす式 T を，式 O から導出する問題と考えることができる．

我々は，この導出を行う系統的な手法として，並行正規表現を等価変換する公理系を使用する．原始的な等価関係を公理として仮定し，公理系の推論規則を用いると，意味は等価だが構文の異なる並行正規表現を導出することができる．我々のアプローチでは，並行オブジェクトモデル O にこれらの推論規則を繰り返し適用することで，関係 $L(O) = L(T)$ を満たす並行スレッドモデル T を導出する．

公理は並行正規表現の原始的な等価関係を表すものである．全ての公理 $X = Y$ について $L(X) = L(Y)$ が成り立つ．表 1 に我々が使用する公理を示す．表 1 の公理はスキーマであり，$A, B$ は任意の並行正規表現，$a, b$ は任意の記号をそれぞれ意味する．また，この表に加えて，正規表現の代数的な性質も公理とみなす．具体的には，正規表現を等価変換する完全な公理系[4] の公理を利用する．

公理系が持つ推論規則は 2 つである．1 つ目は代入規則で，任意の並行正規表現 $P, X, Y$ に対し，公理より $X = Y$ が成立しているならば，$P_{(Y/X)}$ もしくは $P_{(X/Y)}$ を導出してよい．ただし，$P_{(Y/X)}$ は P 中の X の出現を 1 つ，Y で置き換えることを意味する．2 つ目の規則は導出の循環解決規則である．任意の式 X に代入規則を 1 回以上適用した結果，$X.Y + Z$ が導出できる時，$X = Z.Y^*$ を導出してよい．

公理系を用いて変換を行う例は次節以降で示す．また，次節以降，公理以外にも，式の等価関係のスキーマを利用して導出を行う部分がある．これらのスキーマを定理と呼ぶ．本稿で利用する定理の一覧を表 2 に示す．

## 5. 変換例題

この節では 4 節で導入した変換手法を用いて，モデルの変換を行う例題を示す．モデル化を行う対象として，音声と映像を同期再生するマルチメディアプレイヤー仕様の一部を用いる．

| | | | |
|---|---|---|---|
| S01 | $A[\phi]\epsilon$ | = | A |
| S02 | $A[S]B$ | = | $B[S]A$ |
| S03 | $a.A[S]b.B$ | = | $a = b$ のとき $a.(A[S \cup \Sigma_a]B)$<br>$a \neq b, a \notin \Sigma_B \cup S, a \in \Sigma_A \cup S$ のとき<br>$a.(A[S]b.B)$<br>$a \neq b, a \notin \Sigma_B \cup S, b \notin \Sigma_A \cup S$ のとき<br>$a.(A[S]b.B) + b.(a.A[S]B)$<br>$a \neq b, a \in \Sigma_B \cup S, b \in \Sigma_A \cup S$ のとき<br>$\perp$ |
| S04 | $(A + B)[S]C$ | = | $(A[S \cup (\Sigma_B \cap \Sigma_C)]C)$<br>$+(B[S \cup (\Sigma_A \cap \Sigma_C)]C)$ |
| S05 | $A[S]B$ | = | $A[S \cup (\Sigma_A \cap \Sigma_B)]B$<br>$a \notin (\Sigma_A \cup \Sigma_B \cup \{\epsilon, \perp\})$ に対し<br>$A[S \cup \{a\}]B$ |
| S06 | $A[\phi]B$ | = | $\Sigma_A \cap \Sigma_B = \phi$ のとき，$A \parallel B$ |

**表 1** 等価変換のための公理

| | | | |
|---|---|---|---|
| T01 | $A.B[S]C.D$ | = | $(A\|C).(B[S]D)$<br>ただし，<br>$a \in first(B) \cup first(D)$ ならば，<br>$a \in (\Sigma_B \cap \Sigma_D) \cup S$ が成立し，<br>さらに，<br>$(\Sigma_{A.B} \cup S) \cap \Sigma_C = \phi$ かつ<br>$(\Sigma_{C.D} \cup S) \cap \Sigma_A = \phi$ であるとき |
| T02 | $a\|b$ | = | $a.b + a.b$ |
| T03 | $a\|b.B$ | = | $a.(b.B) + b.(a\|B)$ |
| T04 | $a.A\|b$ | = | $(a.A\|b) + b.(a.A)$ |
| T05 | $C[S](A + B)$ | = | $(C[S \cup (\Sigma_B \cap \Sigma_C)]A)$<br>$+(C[S \cup (\Sigma_A \cap \Sigma_C)]B)$ |
| T06 | $(a.A[S]b)$ | = | $a = b$ のとき，$a.A$<br>$a \neq b, a \notin S, b \in \Sigma_A \cup S$ のとき，<br>$a.(A[S]b)$<br>$a \neq b, a \notin S, b \notin \Sigma_A \cup S$ のとき，<br>$a.(A[S]b) + b.a.A$<br>$a \neq b, a \in S, b \in \Sigma_A \cup S$ のとき，$\perp$ |
| T07 | $(a[S]b.B)$ | = | $a = b$ のとき，$b.B$<br>$a \neq b, a \in \Sigma_A \cup S, b \notin S,$ のとき，<br>$b.(a[S]B)$<br>$a \neq b, a \notin \Sigma_A \cup S, b \notin S$ のとき，<br>$b.(a[S]A) + a.b.B$<br>$a \neq b, a \in \Sigma_A \cup S, b \in S$ のとき，$\perp$ |
| T08 | $(a[S]b)$ | = | $a = b$ のとき，$a$<br>$a \neq b, a \notin S, b \notin S$ のとき，<br>$a.b + b.a$<br>それ以外ならば，$\perp$ |

$first(X)$ は $L(X)$ に含まれる記号列の先頭記号の集合を意味する．

**表 2** 等価変換のための定理

### 5.1 仕様とモデル化

本稿で扱うマルチメディアプレイヤーのシステムには，3 つのオブジェクト MAIN,AUDIO,VIDEO が含まれる．表 3 に，ファイルの再生要求が与えられたときの各オブジェクトの振る舞い仕様を示す．なお，この仕様の範囲では，再生するファイルには，音声と映像の

少なくとも一方が含まれることが仮定されている．

MAIN オブジェクトは，次の順序で動作する．
(1) MAIN オブジェクトは，AUDIO オブジェクトと VIDEO オブジェクトに再生すべきデータを渡す．
(2) AUDIO オブジェクトから NO_AUDIO イベントが送信されてきたら，AUDIO オブジェクトからの終了イベントを待って，ダミー音声データを再生する．
(3) VIDEO オブジェクトから NO_VIDEO イベントが送信されてきたら，VIDEO オブジェクトからの終了イベントを待って，ダミー映像データを再生する．
(4) AUDIO オブジェクトと VIDEO オブジェクトの終了を待つ．(2,3 で既に終了したオブジェクトの終了は待たない)

AUDIO オブジェクトは以下の順序で動作を行う．
(1) MAIN オブジェクトから再生すべきデータを受け取る．
(2) 音声データが存在しなければ，VIDEO オブジェクトと MAIN オブジェクトに NO_AUDIO イベントを送って，5 に進む．
(3) NO_VIDEO イベントを受け取ったら，音声の再生を行う．
(4) NO_VIDEO イベントが来なければ，VIDEO オブジェクトと同期しながら音声を再生する．
(5) MAIN に終了イベントを送る．

VIDEO オブジェクトは以下の順序で動作を行う．
(1) MAIN オブジェクトから再生すべきデータを受け取る．
(2) 映像データが存在しなければ，AUDIO オブジェクトと MAIN オブジェクトに NO_VIDEO イベントを送り，5 に進む．
(3) NO_AUDIO イベントを受け取ったら，音声の再生を行う．
(4) NO_AUDIO イベントが来なければ，AUDIO オブジェクトと同期しながら音声を再生する．
(5) MAIN に終了イベントを送る．

**表 3** 振る舞い仕様

この要求仕様を基に，3 つのオブジェクトの振る舞いが以下のように定義できる．

$$\mathtt{MAIN} \equiv \mathtt{C_V}.\mathtt{C_A}.(\mathtt{D_A}.\mathtt{D_V} + \mathtt{N_V}.\mathtt{D_V}.\mathtt{D_P}.\mathtt{D_A} + \mathtt{N_A}.\mathtt{D_A}.\mathtt{D_S}.\mathtt{D_V})$$
$$\mathtt{VIDEO} \equiv \mathtt{C_V}.(\mathtt{S_P} + \mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}$$
$$\mathtt{AUDIO} \equiv \mathtt{C_A}.(\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}$$

各記号が意味する動作を表 4 に示す．図 5 はこれらのオブジェクトを状態遷移図で表現したものである．

| | |
|---|---|
| $\mathtt{C_V}$ | MAIN から VIDEO オブジェクトにデータを送る． |
| $\mathtt{C_A}$ | MAIN から AUDIO オブジェクトにデータを送る． |
| $\mathtt{N_V}$ | NO_VIDEO イベント． |
| $\mathtt{N_A}$ | NO_AUDIO イベント． |
| $\mathtt{S_P}$ | VIDEO と AUDIO オブジェクトで同期してデータを再生する． |
| $\mathtt{D_P}$ | ダミー画像を表示する． |
| $\mathtt{D_S}$ | ダミー音声を再生する． |
| $\mathtt{D_V}$ | VIDEO と MAIN オブジェクト間で交換される終了イベント． |
| $\mathtt{D_A}$ | AUDIO と MAIN オブジェクト間で交換される終了イベント． |

**表 4** 各記号の意味

以上のオブジェクトからなるシステムは，次の並行オブジェクトモデルとして定義できる．

$$\mathtt{MAIN}[\ ]\mathtt{AUDIO}[\ ]\mathtt{VIDEO}$$

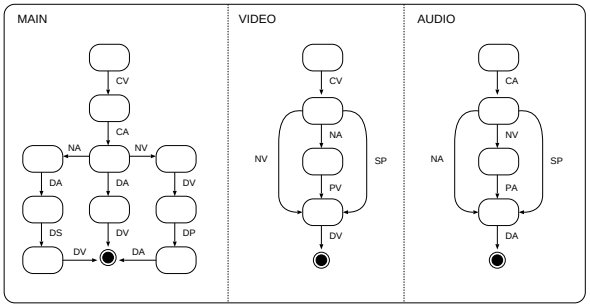



**図 5** オブジェクトの振る舞いモデル

### 5.2 並行スレッドモデルへの変換

この節では，MAIN[ ]VIDEO[ ]AUDIO から，対応する並行スレッドモデルを導出する過程を示す．まず，MAIN[ ](VIDEO[ ]AUDIO) のように演算子が結合しているとみなし，VIDEO[ ]AUDIO を並行スレッドモデル T に変換する．次に MAIN[ ]T と対応する並行スレッドモデルを求める．

以下に，VIDEO[ ]AUDIO と対応する並行スレッドモデルの導出例を示す．なお，導出過程の簡潔さのために，定理を利用している箇所があるが本質的には公理のみで変換できる問題である．また，正規表現の代数的な性質については適用過程を省略している．

$$\mathtt{VIDEO}[\ ]\mathtt{AUDIO}$$
$$\equiv \mathtt{C_V}.(\mathtt{S_P} + \mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}[\ ]\mathtt{C_A}.(\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}$$

定理 T01 を利用して，

$$= (\mathtt{C_V} \,||\, \mathtt{C_A}).(\mathtt{S_P}.\mathtt{D_V} + (\mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}[\ ](\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A})$$

公理 S02,S04 を利用して，

$$= (\mathtt{C_V} \,||\, \mathtt{C_A}).((\mathtt{S_P}.\mathtt{D_V}[\mathtt{N_A}, \mathtt{N_V}](\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}) +$$
$$((\mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}[\mathtt{S_P}](\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}))$$

公理 S02,S04 を利用して，

$$= (\mathtt{C_V} \,||\, \mathtt{C_A}).((\mathtt{S_P}.\mathtt{D_V}[\mathtt{N_A}, \mathtt{N_V}]\mathtt{S_P}.\mathtt{D_A}) +$$
$$(\mathtt{S_P}.\mathtt{D_V}[\mathtt{S_P}, \mathtt{N_A}, \mathtt{N_V}]\mathtt{N_A}.\mathtt{D_A}) +$$
$$(\mathtt{S_P}.\mathtt{D_V}[\mathtt{S_P}, \mathtt{N_A}, \mathtt{N_V}]\mathtt{N_V}.\mathtt{P_A}.\mathtt{D_A})) +$$
$$((\mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}[\mathtt{S_P}](\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}))$$

公理 S03 を利用して，

$$= (\mathtt{C_V} \,||\, \mathtt{C_A}).((\mathtt{S_P}.(\mathtt{D_V}[\mathtt{S_P}, \mathtt{N_A}, \mathtt{N_V}]\mathtt{D_A}) + \bot + \bot) +$$
$$((\mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}[\mathtt{S_P}](\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}))$$

公理 S05 と S06 を利用して，

$$= (\mathtt{C_V} \,||\, \mathtt{C_A}).(\mathtt{S_P}.(\mathtt{D_V} \,||\, \mathtt{D_A}) +$$
$$((\mathtt{N_V} + \mathtt{N_A}.\mathtt{P_V}).\mathtt{D_V}[\mathtt{S_P}](\mathtt{S_P} + \mathtt{N_A} + \mathtt{N_V}.\mathtt{P_A}).\mathtt{D_A}))$$

公理 S03 と S04 を利用して，

$$= (\mathtt{C_V} \,||\, \mathtt{C_A}).(\mathtt{S_P}.(\mathtt{D_V} \,||\, \mathtt{D_A}) +$$
$$((\mathtt{N_V}.\mathtt{D_V}[\mathtt{S_P}, \mathtt{N_A}]\mathtt{S_P}.\mathtt{D_A} + \mathtt{N_A}.\mathtt{D_A} + \mathtt{N_V}.\mathtt{P_A}.\mathtt{D_A})) +$$

$((N_A.P_V.D_V\,[\,S_P,N_V\,]\,S_P.D_A + N_A.D_A + N_V.P_A.D_A)))$

公理 S03 と S04 を利用して,

$= (C_V \,\|\, C_A).(S_P.(D_V \,\|\, D_A) + (N_V.D_V\,[\,S_P,N_A\,]\,N_V.P_A.D_A) +$
$(N_A.P_V.D_V\,[\,S_P,N_V\,]\,N_A.D_A))$

公理 S03 を利用して,

$= (C_V \,\|\, C_A).(S_P.(D_V \,\|\, D_A) + (N_V.(D_V\,[\,S_P,N_A,N_V\,]\,P_A.D_A)) +$
$(N_A.(P_V.D_V\,[\,S_P,N_A,N_V\,]\,D_A))$

公理 S05 と S06 を利用して,

$= (C_V \,\|\, C_A).(S_P.(D_V \,\|\, D_A)+(N_V.(D_V \,\|\, P_A.D_A))+(N_A.(P_V.D_V \,\|\, D_A))$

$\equiv T$

MAIN[ ]T と対応する並行スレッドモデルも同様にして導出可能である．ここでは，導出結果のみ示す.

$C_V.C_A.(S_P.D_A.D_V + N_V.(D_V.D_P \,\|\, P_A).D_A + N_A.(D_A.D_S \,\|\, P_V).D_V)$

図 6 はこの並行スレッドモデルを状態遷移図で表現したものである.

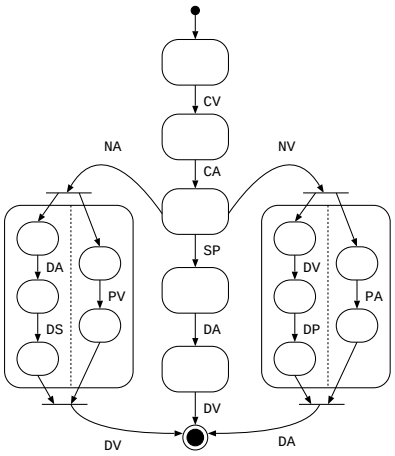



**図 6** 導出された並行スレッドモデル

## 6. 自動変換システム

この節では，前節で示したメディアプレイヤーのモデル変換問題を，計算機を使って自動的に解決した事例を紹介する．本研究で提案する自動変換システムは，推論規則の適用を自動的かつ網羅的に行って，並行スレッドモデルを導出するシステムである．ただし，現在，導出の循環解決規則には未対応である.

### 6.1 自動変換システムの仕様

自動変換システムはオブジェクト 2 つからなる並行オブジェクトモデル X[ ]Y を入力にとり，これと等価な振る舞いを持つ並行スレッドモデルを出力する．並行オブジェクトモデルの変換は，2 つのオブジェクトを単位とした計算に還元できる．このため，このシステムでも 3 つ以上のオブジェクトを含むモデルの変換が可能である．例えば，(X[ ]Y)[ ]Z を計算する場合は，まず X[ ]Y の変換結果を計算する．次にその結果と Z を [ ] 演算子で結合し，再度システムに入力する．このようにして，全体の振る舞いと対応する並行スレッドモデルを獲得することができる.

自動変換システムの実装には C++を用いた．ただし，字句解析のために flex を利用した部分は C で実装されている．コードの規模は約 2500 行 (flex が生成したコードを除く) である．公理および定理はオブジェクトとしてモジュール化されており，定理や公理の追加拡張に柔軟に対応できる．現在，正規表現の代数的性質を表現する公理 11 種と，表 1,2 の公理と定理 14 種がモジュールとして実装されている.

### 6.2 変換戦略

任意の推論規則を適用することによって，等価な振る舞いを持つ並行正規表現を網羅的に探索することができる．しかし，ある振る舞いに対応する並行正規表現は無限に存在するので，やみくもに推論規則を適用すると探索が爆発し，現実的な時間で求める並行スレッドモデルに辿り着くことができない．我々の変換システムでは，探索爆発を防ぐために，以下の対策をとる.

(1) 使用可能な公理の種類を制限する.
(2) 代入規則において代入の方向を制限する.
(3) 探索爆発の防止に役立つ定理を導入する.

これらの対策の具体的な内容は，システムの動作中に適宜切り替えて使用される．本稿では，これらの対策の具体的な内容とその切り替えのタイミングを総じて戦略と呼ぶ．本システムにおける戦略は，逐次的な 9 段階のステップで表現される．以下に各ステップで利用される公理と定理を示す．また，代入方向の制限を示すために記号 $\rightarrow$ を用いる．並行正規表現 P に対し，公理 $X = Y$ と代入規則によって導出される式は $P_{(X/Y)}$ と $P_{(Y/X)}$ の 2 種類がある．以下では，$P_{(Y/X)}$ 側のみを導出するように公理 $X = Y$ を利用することを，特に $X \rightarrow Y$ と表記する.

(1) 定理 T02 $a\|b \rightarrow a.b + a.b$
定理 T03 $a\|b.B \rightarrow a.(b.B) + b.(a\|B)$
定理 T04 $a.A\|b \rightarrow (a.A\|b) + b.(a.A)$
(2) 正規表現の代数則 $A.(B + C) \rightarrow A.B + A.C$
正規表現の代数則 $(A + B).C \rightarrow A.C + B.C$
(3) 公理 S04 $(A+B\,[\,S\,]\,C) \rightarrow (A\,[\,S_1\,]\,C)+(A\,[\,S_2\,]\,B)$

定理 T05 $(A[S]B+C) \to (A[S_1]B)+(A[S_2]C)$

(4) 正規表現の代数則 $(A.B).C \to A.(B.C)$

(5) 公理 S03 $(a.A[S]b.B)$
$\to a.(A[S_1]B)$
$\to a.(A[S_1]b.B)$
$\to a.(A[S_1]b.B)+b.(a.A[S_2]B)$
$\to \bot$

定理 T06 $(a.A[S]b)$
$\to a.A$
$\to a.(A[S_1]b)$
$\to a.(A[S_1]b)+b.a.A$
$\to \bot$

定理 T07 $(a[S]b.B)$
$\to b.B$
$\to b.(a[S_1]B)$
$\to b.(a[S_1]B)+a.b.B$
$\to \bot$

定理 T08 $(a[S]b)$
$\to a$
$\to a.b+b.a$
$\to \bot$

定理 T09 $(a.A[S]b.A) \to b.(a.A[S_1]B)$

(6) 正規表現の代数則 $A.\bot \to \bot$
正規表現の代数則 $A+\bot \to A$

(7) 正規表現の代数則 $A.C+B.C \to (A+B).C$
正規表現の代数則 $A.B+A.C \to A.(B+C)$

(8) 定理 T02 $a.b+b.a \to a\|b$
定理 T03 $a.(b.B)+b.(a\|B) \to a\|b.B$
定理 T04 $(a.A\|b)+b.(a.A) \to a.A\|b$

(9) 正規表現の代数則 $A.C+B.C \to (A+B).C$
正規表現の代数則 $A.B+A.C \to A.(B+C)$

$S_1, S_2$ の内容や公理の適用条件は省略する．公理と定理の定義 (表 1,2) を参照されたい．

この戦略の各ステップは，式に代入規則を適用可能な部分がある限り繰り返し実行される．代入規則を適用できる部分がなくなったら，戦略は，次のステップに移行する．ただし，ステップ 7〜9 では，推論規則が適用できなくなった場合，以下の副戦略が順番に実行される．

(1) $A\|B = B\|A$

(2) $(A.B).C = A.(B.C)$

(3) $A+(B+C) = (A+B)+C$

(4) $A+B = B+A$

この副戦略中のどこかで代入規則の適用に成功した場合，即座に副戦略は中断される．そして，元の戦略のステップに戻って，副戦略で導出された式を対象に，公理の適用が再開される．もし，副戦略側でも一切代入規則の適用ができなかった場合，戦略は次のステップに移行する．また，副戦略においては，導出された式の履歴が作成され，既に履歴に存在する式の導出は抑止される．ただし，この履歴は元の戦略が次のステップに進む際に破棄される．

### 6.3 戦略の直感的意味

今回，我々の導入した戦略は，手作業で各種の変換問題を解いた経験に基づいて作成されたものである．直感的には，戦略の各ステップは以下のような式の書き換え作業に相当する．

- **ステップ 1〜3:** $[S]$ 演算子を分配し，起りうる全ての同期複合の可能性を網羅する．例えば，$(A+B)[S](C+D)$ なる型の式は，$(A[S_1]C)+(B[S_1]C)+(B[S_3]C)+(B[S_4]D)$ 型に展開される．
- **ステップ 4〜6:** $[S]$ 演算子の両辺から，最も左端の記号を順に取り出す．取り出した記号が同期通信を意味するなら，その部分式の左端にくくり出す．例えば $a.b[\phi]a.c$ は $a.(b[a]c)$ と書き換えられる．並行に行われる動作を意味するなら，インターリーブした記号列に展開する．同期通信に失敗する部分は式から削除される．
- **ステップ 7〜9:** 並行動作をインターリーブに展開する規則を逆向きに適用し，式から並行動作に相当する部分を再抽出する．抽出した範囲は $\|$ 演算子を使った表現で置き換える．

### 6.4 定理による探索爆発の抑止

例として，戦略のステップ 3 について考える．このステップで行いたいことは $A+B[S]C$ 型の式を $A[S_1]C+B[S_2]C$ に展開することと，$C[S]A+B$ 型の式を $C[S_1]A+C[S_2]B$ 型に展開することである．実際，この操作は公理 S04 と公理 S02 の組合わせで達成できる．公理 S04 の左辺は $A+B[S]C$ 型であるので，この公理を直接 $C[S]A+B$ 型の式に適用することはできない．しかし，公理 S02 $A[S]B = B[S]A$ を利用すると，$A+B[S]C$ 型の式が得られる．したがって，公理 S04 の適用が可能となる．しかし，公理 S04 は $C[S]A+B$ だけでなく，$A[S]B$ 型のどんな部分にも適用できてしまう．このため，公理 S04 の使用を認めると，導出される式の範囲が広がり，探索爆発につながる可能性がある．定理 T05 は，このような導出可能な式の範囲を制限する意図で導入されている．その他に戦略で利用されている定理も，全て公理の組合わせで実現できるが，定理 T05 と同様に探索爆発を防ぐ

目的で導入されている．

### 6.5 変換結果

本システムに，メディアプレイヤーの例題を与えた結果，4 節に示した変換結果と一致する並行スレッドモデルの自動的な導出に成功した．各ステップおよび全体での推論規則適用回数を表 5,6 に示す．表 5 は $\mathtt{T} \equiv \mathtt{VIDEO}[\ ]\mathtt{AUDIO}$ を求めた後，$\mathtt{MAIN}[\ ]\mathtt{T}$ を計算したものである．表 6 は $\mathtt{T}' \equiv \mathtt{MAIN}[\ ]\mathtt{VIDEO}$ を導出した後，$\mathtt{T}'[\ ]\mathtt{AUDIO}$ を計算した結果である．

| ステップ | $\mathtt{MAIN}[\ ]\mathtt{T}$ | $\mathtt{T} \equiv \mathtt{VIDEO}[\ ]\mathtt{AUDIO}$ |
|---|---|---|
| 1 | 11 | 0 |
| 2 | 52 | 15 |
| 3 | 94 | 14 |
| 4 | 49 | 3 |
| 5 | 127 | 25 |
| 6 | 90 | 0 |
| 7 | 1110 | 2180 |
| 8 | 122 | 210 |
| 9 | 64 | 222 |
| 計 | 1719 | 2669 |

**表 5** 推論規則の適用回数 1

| ステップ | $\mathtt{T}'[\ ]\mathtt{AUDIO}$ | $\mathtt{T}' \equiv \mathtt{MAIN}[\ ]\mathtt{VIDEO}$ |
|---|---|---|
| 1 | 7 | 0 |
| 2 | 42 | 15 |
| 3 | 40 | 16 |
| 4 | 44 | 4 |
| 5 | 74 | 52 |
| 6 | 45 | 28 |
| 7 | 8562 | 2578 |
| 8 | 81 | 221 |
| 9 | 54 | 156 |
| 計 | 8949 | 3070 |

**表 6** 推論規則の適用回数 2

## 7. 考　　察

我々が用いた戦略では，ステップ 7 が最も多く推論規則の適用を必要としている．その適用回数のほとんどは，副戦略によって網羅的に式の導出が発生した回数である．ステップ 6 の終了時点で導出されている式は，可能な振る舞いの列を全て + 演算子で繋いだ式である．この式は，自動変換システム実行中に生成される式の中で最も長い．副戦略においては，式が長いほど導出可能な式の個数が多くなる．したがって，ステップ 7 で行われる推論規則の適用回数は他のステップに比べて多くなる．

また，ステップ 7 における推論規則の適用回数は，ステップ 6 終了時に導出されている式の形に強く依存する．表 6 の $\mathtt{T}'[\ ]\mathtt{VIDEO}$ と表 5 の $\mathtt{MAIN}[\ ]\mathtt{T}$ のステップ 7 を比べると，表 6 における適用回数の方が圧倒的に多い．ステップ 6 で導出された式の構造的な違いが原因であった．ステップ 6 で導出される式は，可能な振る舞いの全ての列が + 演算子で結合されたものである．ここで，表 $\mathtt{T}'[\ ]\mathtt{VIDEO}$ と $\mathtt{MAIN}[\ ]\mathtt{T}$ が表現する振る舞いが等しいことを考えると，ステップ 6 で導出された式の違いは，式の括弧の付き方と，+ 演算子で結合された部分式の出現順序の違いである．しかし，このようにごく単純な式の違いであっても，ステップ 7 で導出される式の順番は大きく影響を受ける．ステップ 7 は連続的に導出を行って，適当な式に辿り着いた段階終了するので，導出される式の順序によっては終了までに非常に多く推論規則の適用を要することがある．

今後の課題として，ステップ 7 における規則の適用回数を削減する定理や戦略を与えることが重要である．

## 8. ま と め

本稿では，並行正規表現とその公理系を用いて並行オブジェクトモデルを並行スレッドモデルに変換する方法について述べた．また，この変換を自動化するシステムを提案した．さらに，メディアプレイヤーのモデル変換問題を処理し，自動変換に成功した事例を紹介した．以下に今後の課題を示す．

- 自動変換の効率化
- 循環解決規則に対応した自動変換システムの実装
- さらなる例題システムへの適用

## 参 考 文 献


1) Vijay K. Garg, M.T. Ragunath: *Concurrent regular expressions and their relationship to Petri nets*, Theoretical Computer Science 96, pp.285-304, 1992.
2) Maher Awad, Juha Kuusela and Jurgen Ziegler : *Object-Oriented Technology for Real-Time Systems* , Prentice Hall, 1996.
3) 青木利晃, 片山卓也: オブジェクト指向組み込みシステム開発のための設計モデル SES モデル, 日本ソフトウェア科学会 FOSE2000, pp.157-164, 2000.
4) Arto Salomma : *Two Complete Axiom Systems for the Algebra of Regular Events* , Journal of the Association for Computing Machinery, Vol.13, No. 1, pp158-169, 1966.
5) M.Okazaki, T.Aoki, T.Katayama: *Extracting threads from concurrent objects for the design of embedded systems*, Proceedings of Asia-Pacific Software Engineering Conference APSEC 2002, pp.107-116, 2002.